

Tous documents interdits.

NOM :

PRÉNOM :

▷ **Question 1.** (2 pt) Indiquer si l'affirmation est correcte ou non :

Vrai	Faux	
<input type="checkbox"/>	<input type="checkbox"/>	Il est toujours possible d'appeler n'importe quelle méthode d'un objet à partir du corps d'une méthode d'un autre objet.
<input type="checkbox"/>	<input type="checkbox"/>	Il peut exister plusieurs instances d'une classe abstraite (déclarée <code>abstract</code>).
<input type="checkbox"/>	<input type="checkbox"/>	Le receveur d'un appel de méthode n'est jamais de type primitif.
<input type="checkbox"/>	<input type="checkbox"/>	Un champ est marqué <code>static</code> lorsque sa valeur ne doit pas changer au cours de l'exécution du programme.
<input type="checkbox"/>	<input type="checkbox"/>	Une sous-classe d'une classe concrète (non abstraite) est toujours concrète.
<input type="checkbox"/>	<input type="checkbox"/>	Une sous-classe d'une classe abstraite peut être abstraite.
<input type="checkbox"/>	<input type="checkbox"/>	Une classe déclarée <code>public</code> doit avoir tout ses champs déclarés <code>public</code> .
<input type="checkbox"/>	<input type="checkbox"/>	Il est toujours possible de transtyper une référence dans le type <code>Object</code> .

▷ **Question 2.** (2 pt)

Parmi les déclarations de classe et d'interface suivantes, indiquer lesquelles sont valides (certaines déclarations seront ré-utilisées par les déclarations qui les suivent) :

- | | |
|--|---|
| (a) <input type="checkbox"/> <code>class Foo { }</code> | (h) <input type="checkbox"/> <code>interface Zoo extends Foo { }</code> |
| (b) <input type="checkbox"/> <code>class Bar implements Foo { }</code> | (i) <input type="checkbox"/> <code>interface Boo extends Fi { }</code> |
| (c) <input type="checkbox"/> <code>interface Baz { }</code> | (j) <input type="checkbox"/> <code>class Zoom implements Fi, Baz { }</code> |
| (d) <input type="checkbox"/> <code>interface Fi { }</code> | (k) <input type="checkbox"/> <code>class Toon extends Foo, Zoom { }</code> |
| (e) <input type="checkbox"/> <code>interface Fee implements Baz { }</code> | (l) <input type="checkbox"/> <code>interface Vroom extends Fi, Baz { }</code> |
| (f) <input type="checkbox"/> <code>interface Zee implements Foo { }</code> | (m) <input type="checkbox"/> <code>class Yow extends Foo implements Fi { }</code> |

▷ **Question 3.** (1 pt)

Considérer les classes suivantes et indiquer **la réponse correcte** :

```

1 class Shape {
2     private String color;
3
4     public Shape(String color) {
5         System.out.print("Shape");
6         this.color = color;
7     }
8 }
9
10 class Rectangle extends Shape {
11     public Rectangle() {
12         super("bleu");
13         System.out.print("Rectangle");
14     }
15 }
16
17 public class TestConstructor {
18     public static void main(String[] args) {
19         new Rectangle();
20     }
21 }

```

- Ce code ne compile pas (erreur ligne 5).
- Ce code ne compile pas (erreur ligne 12).
- Ce code compile et le programme affiche : `Shape`
- Ce code compile et le programme affiche : `Rectangle`
- Ce code compile et le programme affiche : `ShapeRectangle`
- Ce code compile et le programme affiche : `RectangleShape`

▷ **Question 4.** (2 pt)

Lors de la surcharge d'une méthode (*overloading*) dans une classe et non pas de la redéfinition de celle-ci (*overriding*) dans une sous-classe :

	PEUT	DOIT	NE DOIT PAS
La nouvelle définition [...] changer la liste des paramètres (type ou nombre de paramètres).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
La nouvelle définition [...] changer le type de la valeur de retour.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
La nouvelle définition [...] changer le modificateur d'accès (private, public, protected)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
La nouvelle définition [...] lancer de nouvelles exceptions.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

▷ **Question 5.** (2 pt)

Considérer les deux classes **Geek** et **Sheldon** où la méthode **say()** a été redéfinie (*overriding*). Indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 14 :

<pre> 1 class Geek { 2 public void say() { 3 System.out.println("hello!"); 4 } 5 } 6 class Sheldon extends Geek { 7 public void say() { 8 System.out.println("bazinga!"); 9 } 10 public static void main(String[] args) { 11 Geek g = new Sheldon(); 12 Sheldon s = new Sheldon(); 13 Geek g2 = new Geek(); 14 /* à remplacer */ 15 } 16 } </pre>	<pre> g.say(); g2.say(); s.say(); ((Sheldon) g).say(); ((Sheldon) g2).say(); ((Geek) s).say(); </pre>	<table border="1"> <thead> <tr> <th>Erreur (compilation)</th> <th>Affiche hello!</th> <th>Affiche bazinga!</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> <tr> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> <tr> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> <tr> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> <tr> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> </tbody> </table>	Erreur (compilation)	Affiche hello!	Affiche bazinga!	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Erreur (compilation)	Affiche hello!	Affiche bazinga!																		
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																		
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																		
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																		
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																		
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																		

▷ **Question 6.** (5 pt)

Considérer les classes suivantes et réaliser des schémas de la mémoire (états de la pile et du tas) lors de l'exécution de la méthode **main()** de la classe principale **mem.Main** aux points identifiés dans le code source.

Vous préciserez l'affichage obtenu sur la sortie standard pour l'ensemble de l'exécution.

<pre> 1 package mem; 2 3 public class Geek { 4 private int qi; 5 private String name; 6 7 public Geek(String n, int iq) { 8 name = n; 9 qi = iq; 10 } 11 public int getQI() { 12 return qi; 13 } 14 public void learn(int p) { 15 qi += p; 16 } 17 public void setName(String n) { 18 name = n; 19 } 20 public Geek clone(int p) { 21 return new Geek(name, qi + p); 22 } 23 public boolean eq(Geek o) { 24 return (name == o.name && qi == o.qi); 25 } 26 public String toString() { 27 return "Geek[" + name + "," + qi + "];" 28 } 29 } </pre>	<pre> 1 package mem; 2 3 public class Main { 4 public static void main(String[] args) { 5 Geek t1 = new Geek("Leonard", 173); 6 Geek t2 = t1.clone(14); 7 t2.setName("Sheldon"); 8 GeeksGroup tg = new GeeksGroup(); 9 tg.join(t1); 10 tg.join(t2); 11 12 System.out.println("POINT 1"); 13 System.out.println(t1); 14 System.out.println(t2); 15 System.out.println(tg); 16 // POINT 1 17 18 GeeksGroup bbt = tg.duplicate(); 19 20 System.out.println("POINT 2"); 21 System.out.println(bbt.getGeek(0)); 22 System.out.println(bbt.getGeek(1)); 23 System.out.println(tg); 24 System.out.println(bbt); 25 System.out.println("same: "+(bbt == tg)); 26 System.out.println("eq: "+bbt.eq(tg)); 27 // POINT 2 28 29 30 Geek t3 = new Geek("Howard",109); 31 tg.join(t3); 32 bbt.join(t3); 33 34 System.out.println("POINT 3"); 35 System.out.println(tg); 36 System.out.println(bbt); 37 System.out.println("same: "+(bbt == tg)); 38 System.out.println("eq: "+bbt.eq(tg)); 39 // POINT3 40 41 Geek t4 = new Geek("Rajesh", 108); 42 tg.join(t4); 43 bbt.join(t4.clone(0)); 44 45 System.out.println("POINT 4"); 46 System.out.println(tg); 47 System.out.println(bbt); 48 System.out.println("same: "+(bbt == tg)); 49 System.out.println("eq: "+bbt.eq(tg)); 50 // POINT4 51 52 bbt.study(10); 53 System.out.println("POINT 5"); 54 System.out.println(tg); 55 System.out.println(bbt); 56 System.out.println("same: "+(bbt == tg)); 57 System.out.println("eq: "+bbt.eq(tg)); 58 // POINT5 59 } 60 } </pre>
<pre> 1 package mem; 2 3 public class GeeksGroup { 4 private Geek[] geeks = new Geek[10]; 5 private int size = 0; 6 7 public void join(Geek t) { 8 geeks[size] = t; 9 size++; 10 } 11 public Geek getGeek(int i) { 12 return geeks[i]; 13 } 14 public void study(int iqi) { 15 for (int i = 0; i < size; i++) 16 geeks[i].learn(iqi); 17 } 18 public GeeksGroup duplicate() { 19 GeeksGroup g = new GeeksGroup(); 20 g.size = size; 21 for (int i = 0; i < size; i++) 22 g.geeks[i] = geeks[i]; 23 return g; 24 } 25 public boolean eq(GeeksGroup g) { 26 for (int i = 0; i < size && i < g.size; i++) { 27 if (geeks[i] != g.geeks[i]) 28 return false; 29 } 30 return true; 31 } 32 public String toString() { 33 String s = "Geeks(" + size; 34 for (int i = 0; i < size; i++) 35 s += "," + geeks[i].toString(); 36 s += ")"; 37 return s; 38 } 39 } </pre>	

▷ **Question 7.** (1 pt)

Indiquer 3 raisons (il y en a beaucoup plus) faisant que la méthode `eq(Geek o)` de la classe `mem.Geek` n'implémente pas correctement le contrat habituel de la méthode `equals()` définie dans la classe `Object` (méthode que l'on doit généralement redéfinir dans toute nouvelle classe).

▷ **Question 8.** (3 pt)

Écrire le code de la méthode `maxQI()` de la classe `mem.GeeksGroup` qui retourne une référence vers un objet de classe `mem.Geek`. L'objet retourné correspond au `Geek` possédant le Q.I. le plus élevé. Dans le cas où il y aurait plusieurs `Geek` avec le même Q.I. une exception de type `TooManyCleverGeekException` devra être levée.

▷ **Question 9.** (2 pt)

L'implémentation actuelle de la classe `mem.GeeksGroup` ne supporte pas l'ajout de plus de 10 membres au sein d'un groupe. Proposer une solution permettant de supprimer cette limitation. Écrire le code de cette nouvelle implémentation (penser à bien regarder l'impact de vos modifications sur toutes les méthodes de la classe).